Chapitre 1

INTRODUCTION

1. Introduction au Génie Logiciel

1. 1. Les logiciels et le Génie Logiciel

La non qualité des systèmes informatiques, a des conséquences qui peuvent être graves, voire désastreuses. Citons quelques conséquences de bogues (bugs):

- Mission Vénus : passage à 5000 000 km de la planète au lieu des 5000 km prévus.

Cause : remplacement d'une virgule par un point.

- Perte de satellites dans les années 70.

Cause : +I au lieu de +1 dans une boucle du programme source Fortran.

- Hôpital : Décès de patients.

Cause : erreur dans les programmes de monitoring.

- Premier lancement d'Ariane V : elle explose en vol.

Cause : logiciel de plate-forme inertielle tel quel d'Ariane IV sans nouvelle validation. Le logiciel a jugé l'inclinaison d'Ariane V non conforme au plan de tir (d'Ariane IV) et a lancé l'ordre de destruction. En fait, tout était normal !

En 1995, une étude du Standish Group (http://www.standishgroup.com) dressa un tableau accablant de la conduite des projets informatiques. Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8380 applications, cette étude a pu établir que :
– 16, 2% seulement des projets étaient conformes aux prévisions initiales,
– 52,7% avaient subi des dépassements en coût et en délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes.
– 31, 1% ont été purement abandonnés durant leur développement.
Pour les grandes entreprises (qui lancent proportionnellement davantage de gros projets), le taux de succès est de 9% seulement, 37% des projets sont arrêtés en cours de réalisation, 50% aboutissent hors délai et hors budget.

On considère donc que la prise de conscience d'une crise du logiciel date de la fin des années 1960.
Cette prise de conscience eut lieu à Garmisch (Allemagne) en 1968 lors de la conférence internationale sur la conception de logiciels.
L'idée était d'améliorer la qualité des logiciels et ce en adoptant des méthodes de développement et en les respectant. De ce fait, est née une discipline appelée '**Génie Logiciel'**.

Qu'est ce que le Génie Logiciel ?

- Selon le standard IEEE 610.12 :
  "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software that is the application of engineering to software"

  Ce qu se traduit par:
  « L'application d'une approche systématique, maîtrisée, quantifiable au développement, à l'exploitation et à la maintenance du logiciel : c'est-à-dire l'application de l'ingénierie au logiciel »

- Discipline (= méthodes, techniques et outils)
  - basée sur le savoir (théorique)
  - le savoir-faire (pragmatique)
  - et le faire savoir (communication)

- pour produire (développement)
- de façon industrielle (taille, diffusion)
- des logiciels (= les produits)
- *de qualité au meilleur prix …*

Le Génie Logiciel, noté souvent GL, est donc l'ensemble des activités de conception et de mise en oeuvre des procédures tendant à rationaliser la production du logiciel et son suivi. Autrement dit :
LE GL se préoccupe des procédés de fabrication des logiciels en s'assurant que les quatre critères suivants soient respectés : (**Règles du CQFD = Coùt Qualité Fonctionnalité Délai**)
- le logiciel proposera les **F**onctionnalités désirées et répondra aux besoins des utilisateurs
- le logiciel sera de **Q**ualité
- les **C**oûts resteront dans les limites prévues initialement
- les **D**élais de livraison ne seront pas dépassés.

Facteurs de qualité :
Validité : aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.
Fiabilité ou robustesse : aptitude d'un produit logiciel à fonctionner dans des conditions anormales.
Extensibilité (maintenance): facilité avec laquelle un logiciel se prête à sa maintenance, c'est-à-dire à une modification ou à une extension des fonctions qui lui sont demandées.
Réutilisabilité : aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.
Compatibilité : facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.
Efficacité : Utilisation optimales des ressources matérielles.
Portabilité : facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.
Vérifiabilité : facilité de préparation des procédures de test.
Intégrité : aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisés.
Utilisabilité : facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

Ces facteurs sont parfois contradictoires, le choix des compromis doit s'effectuer en fonction du contexte.

1. 2. Concepts du GL

1. 2. 1. Participants et Rôles

Le développement d'un système logiciel nécessite la collaboration de plusieurs personnes :
Participants*:*
- Clients
- Développeurs
- Gestionnaire de projets
- Utilisateurs
- Rédacteur technique

Un ensemble de tâches (responsabilités) associées à un participant est appelé **Rôle.**
Plusieurs rôles peuvent être attribués à un même participant.
Exemple: fournir les besoins de haut niveau: client,

fournir les connaissances du domaine: utilisateur,
construire le système (spécifier, concevoir, implémenter, tester): développeur.

1. 2. 2. Système et modèle de système

Un système est un ensemble d'éléments en interaction dynamique organisés en fonction d'un but. Un système peut être abstrait ou concret, naturel ou artificiel. On parle ainsi du système solaire, du système de sécurité sociale, d'un système informatique…etc.
Les systèmes sont souvent imbriqués, un système contient d'autres systèmes, appelés des sous-systèmes, et lui-même est contenu dans un système plus grand qui constitue son environnement.

Caractéristiques :

Un système est caractérisé par :

Sa structure : les éléments qui le composent.
Son évolution : les états successifs par lesquels il passe.
Les fonctions : ce qu'il sait faire.

Dynamique :

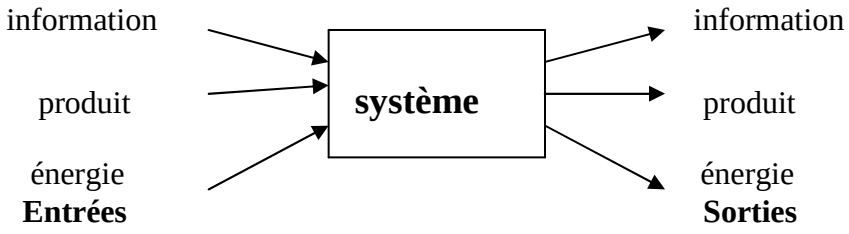
Un système reçoit des données d'entrée d'autres systèmes ou de l'environnement. Les entrées subissent des modifications (transformations). Les transformations produisent des sorties qu'absorbent d'autres systèmes ou l'environnement.

Remarque :

Quand on applique le concept de système à la définition et à la résolution des problèmes, on parle alors *d'**approche systématique.***
Quelques systèmes actuels :

- Gros systèmes de gestion (systèmes d'information); le plus souvent des systèmes transactionnels construits autour d'une base de donnée centrale ;
- Systèmes temps réel, qui doivent répondre à des événements dans des limites de temps prédéfinies et strictes ;
- Systèmes distribués sur un réseau de machines (distribution des données et/ou des traitements), 'nouvelles architectures' liées à Internet;
- Systèmes embarqués et systèmes critiques, interfacés avec un système à contrôler (ex: aéronautique, centrales nucléaires, ...).

Un **modèle** est une abstraction (représentation) d'un système. Il permet de comprendre et de gérer le comportement d'un système.
L'utilisation de modèles prend une part de plus en plus importante dans les projets technologiques menés par les ingénieurs, que ce soit pour la définition de systèmes, leur conception ou leur réalisation.



1. 2. 3. Activités, tâches et Ressources

- Une **activité** est un ensemble de tâches à exécuter pour un propos spécifique.
  Exemple : Elicitation des besoins.

- Une **tâche** représente une unité atomique de travail. Elle utilise des **ressources** et peut dépendre d'autres tâches.
  Exemple: tester le distributeur de tickets

- Une **ressource** est un moyen (matériel, logiciel ou humain) utilisé pour accomplir un travail.
  Exemple: base de données des tarifs de tickets

1. 2. 4. Besoins fonctionnels et non fonctionnels

- Un **besoin fonctionnel** est la spécification d'une fonction que le système doit supporter.

Exemple: l'utilisateur doit pouvoir accéder aux informations2121 de tarification des tickets.

- Un **besoin non fonctionnel** est une contrainte sur l'opération du système non liée directement à la fonction.

Exemple: absence d'interblocage, sécurité.

1. 2. 5. Notations, méthodes et méthodologies

- Une **notation** est un ensemble de règles graphiques ou textuelles pour représenter un modèle.

Exemple: notation UML.

- Une **méthode** est une technique qui spécifie les étapes de résolution d'un problème spécifique.

Exemples: recette de cuisine, algorithme de tri, gestion de configuration de système.

- Une **méthodologie** est une collection de méthodes pour résoudre une classe de problèmes. Elle spécifie comment et quand chaque méthode doit être utilisée.

Exemple: méthodologie orientée objet pour développer un logiciel.

1. 3. Cycle de vie d'un logiciel

Le cycle de vie d'un logiciel (en anglais software life cycle), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.
L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend généralement au minimum les étapes suivantes :
Définition des objectifs : Cette étape consiste à définir la finalité du projet et son inscription dans une stratégie globale.
Analyse des besoins et faisabilité : c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes puis l'estimation de la faisabilité de ces besoins.
Spécification ou conception générale : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.
Conception détaillée : Cette étape consiste à définir précisément chaque sous-ensemble du logiciel.
Codage (Implémentation ou programmation) : c'est la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
Tests unitaires : Ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.
Intégration : L'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de tests d'intégration consignés dans un document.
Qualification (ou recette) : C'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.
Documentation : Elle vise à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs.
Mise en production
Maintenance : Elle comprend toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

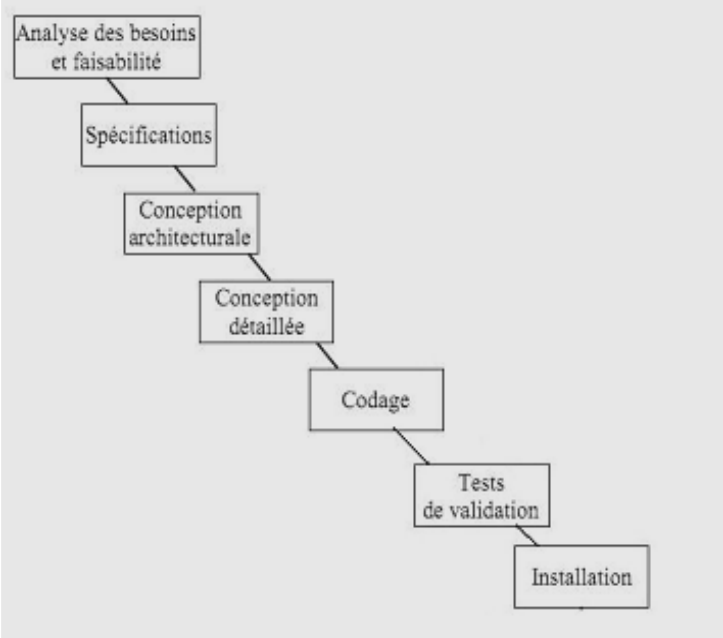
La séquence et la présence de chacune de ces activités dans le cycle de vie, dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement. Le cycle de vie permet de prendre en compte, en plus des aspects techniques, l'organisation et les aspects humains.

Modèles de cycle de vie d'un logiciel :

Cycle en Cascade :

Ce cycle est hérité du bâtiment. Ce modèle repose sur les hypothèses suivantes:

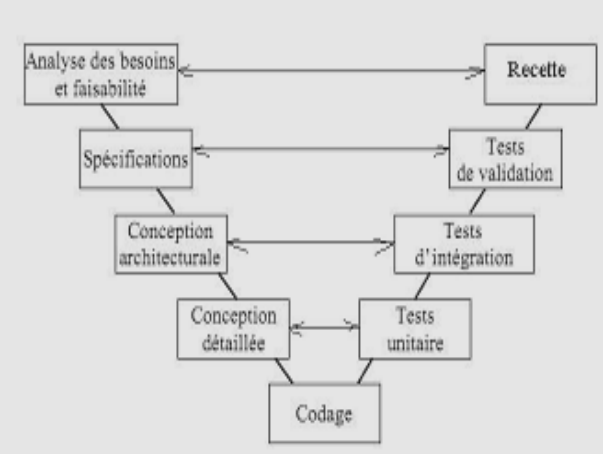
- On ne peut pas construire la toiture avant les fondations.
- Les conséquences d'une modification en amont du cycle ont un impact majeur sur les coûts en aval.



Le modèle original ne comportait pas de possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, ce qui, dans la pratique, s'avère insuffisant.

Cycle en V :

Imaginé pour pallier le problème de réactivité du modèle en cascade, ce modèle permet en cas d'anomalie, de limiter un retour aux étapes précédentes. Les phases de la partie montante doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le logiciel. Il met en évidence la nécessité d'anticiper et de préparer dans les étapes descendantes les « attendus » des futures étapes montantes.
Le cycle en V est devenu un standard de l'industrie du développement de logiciel et de la gestion de projet depuis les années 1980.

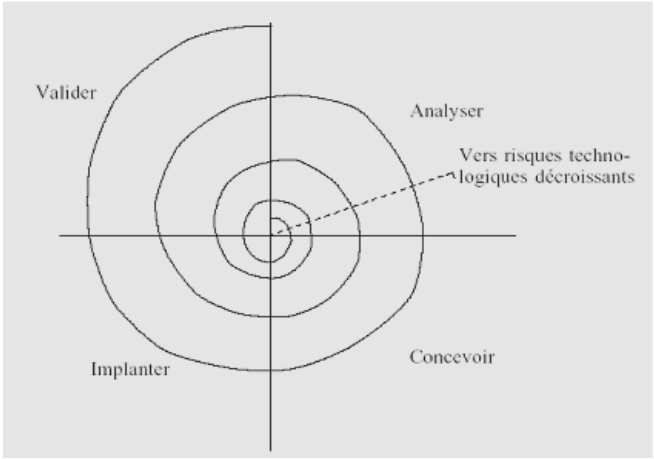


Cycle en spirale :

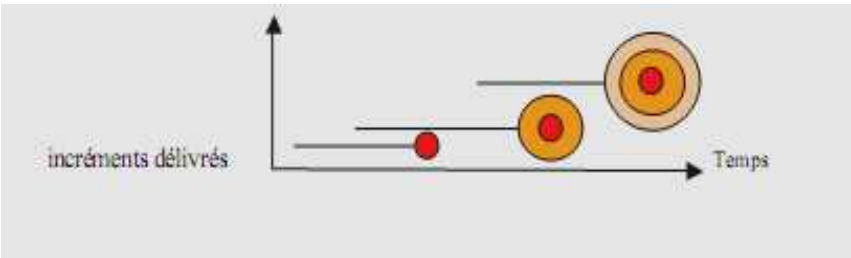
Ce modèle s'appuie sur une succession de cycles dont chacun se déroule en quatre phases :

- Analyse initiale des besoins et des objectifs du cycle (solutions et contraintes) ou analyse à partir du cycle précédant,
- Etude des risques, évaluation des solutions de remplacement et éventuellement conception,
- Développement et vérification de la solution résultant de l'étape précédente,
- Examen du produit et projection vers le cycle suivant.

Le modèle en spirale (cher!) est peu utilisé en pratique vue la difficulté de le mettre en œuvre.



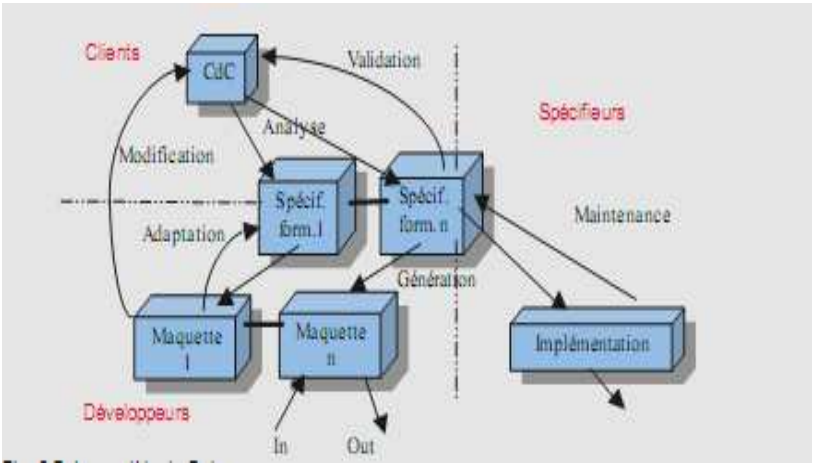
Modèle incrémental :



- **Principe**: un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents
- **Avantages** de ce type de modèle:
  - chaque développement est moins complexe ;
  - les intégrations sont progressives; il est ainsi possible de livrer et de mettre en service chaque incrément ;
  - il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.
- **Risques** de ce type de modèle:
  - remettre en cause les incréments précédents ou pire le noyau ;
  - ne pas pouvoir intégrer de nouveaux incréments.

Modèle de Balzer :

Le modèle de Balzer associe développement incrémental et utilisation de spécifications formalisées, elles même développées de manière incrémentale et maintenues



Modèle idéal ?

Il n'y a pas de modèle idéal car tout dépend des circonstances. Le modèle en cascade ou en V est risqué pour les développements innovants car les spécifications et la conception risquent d'être inadéquats et souvent remis en cause. Le modèle incrémental est risqué car il ne donne pas beaucoup de visibilité sur le processus complet. Le modèle de Balzer est risqué car il exige des spécialistes de très haut niveau. Le modèle en spirale est un canevas plus général qui inclut l'évaluation des risques.

Souvent, un même projet peut mêler différentes approches, comme le prototypage pour les sous-systèmes à haut risque et la cascade pour les sous systèmes bien connus et à faible risque.

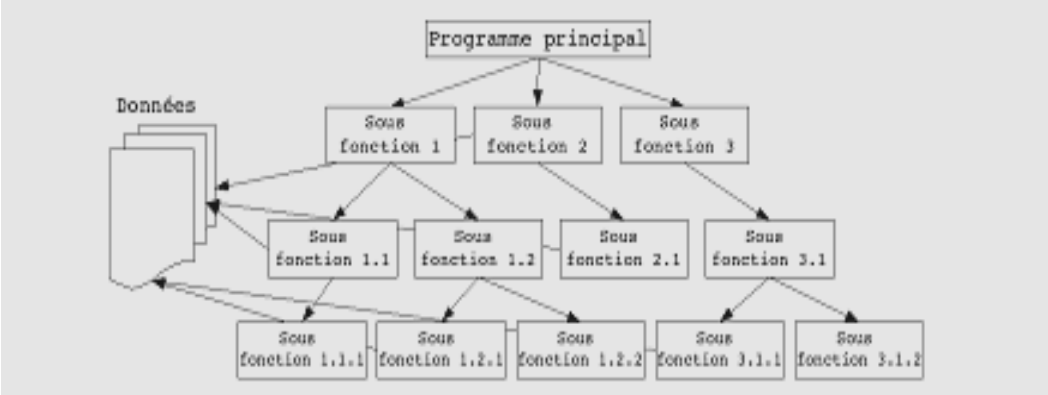
1. 4. Méthodes d'analyse et de conception de logiciels

Les méthodes d'analyse et de conception fournissent une méthodologie et des notations standard qui aident à concevoir des logiciels de qualité. Il existe différentes manières pour classer ces méthodes, dont :

- La distinction entre composition et décomposition : Elle met en opposition d'une part les méthodes ascendantes qui consistent à construire un logiciel par composition à partir de modules existants et d'autre part, les méthodes descendantes qui décomposent récursivement le système jusqu'à arriver à des modules programmables simplement.
- La distinction entre fonctionnel (dirigée par le traitement) et orientée objet

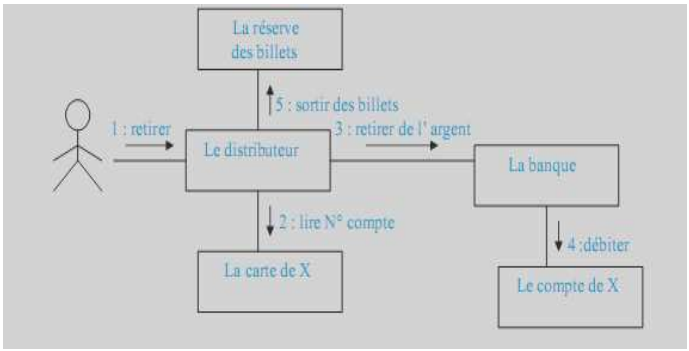
Méthodes fonctionnelles ou structurées:

C'et une approche hiérarchique descendante et modulaire qui dissocie le problème de la représentation des données, du problème du traitement de ces données (Exemple: SADT (Structured Analysis Design Technique). Une modification des données entraîne une modification d'un nombre important de fonctions éparpillées et difficiles à identifier dans la hiérarchie de cette décomposition.



L'approche orientée objet:

Cette approche considère le logiciel comme une collection d'objets dissociés, et identifiés (identité, attributs, méthodes) qui interagissent. Contrairement à l'approche fonctionnelle, elle rapproche les données et leurs traitements associés. Elle conduit alors, à des architectures logicielles fondées sur les objets du système, plutôt que sur la fonction qu'il est censé réaliser. Exemple :



Ainsi la technologie objet est la conséquence ultime de la modularisation du logiciel, démarche qui vise à maîtriser sa production et son évolution. Mais malgré cette continuité logique les langages objet ont apporté en pratique un profond changement dans l'art de la programmation : ils impliquent en effet un changement de l'attitude mentale du programmeur.

Concepts importants de l'approche objet

Certains concepts importants sont spécifiques à cette approche et participent à la qualité du logiciel

Notion de classe

Une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à toute une famille d'objets et permettant de créer (instancier) des objets possédant ces propriétés. Les autres concepts importants qu'il nous faut maintenant introduire sont l'encapsulation, l'héritage et l'agrégation.

Encapsulation

L'encapsulation consiste à masquer les détails d'implémentation d'un objet en définissant une interface. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet. L'encapsulation

- facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.
- garantit l'intégrité des données, car elle permet d'interdire, ou de restreindre, l'accès direct aux attributs des objets.

Héritage, Spécialisation, Généralisation et polymorphisme

L'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe. Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.

Ainsi, la spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple. L'héritage évite la duplication et encourage la réutilisation.

Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la généricité, et donc la qualité du code.

L'agrégation consiste en une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe. Une relation d'agrégation permet donc de définir des objets composés d'autres objets. L'agrégation permet donc d'assembler des objets de base, afin de construire des objets plus complexes.

2. Introduction à la modélisation orientée objet

2. 1. Modèle et modélisation

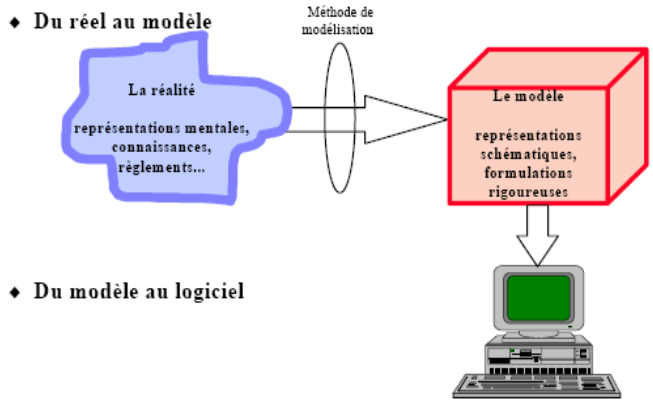
Qu'est-ce qu'un modèle ?

Un modèle est une représentation abstraite et simplifiée (i.e. qui exclut certains détails), d'une entité (Phénomène, processus, système, etc.) du monde réel en vue de le décrire, de l'expliquer ou de le prévoir. Il est centré sur la représentation conceptuelle et physique d'un système.

La modélisation est alors une technique d'ingénierie permettant de représenter un système; elle s'appuie sur l'établissement de modèles.

Pourquoi modéliser ?

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence. Un modèle est un langage commun, précis, qui est connu par tous les membres de l'équipe et il est donc, à ce titre, un vecteur privilégié pour communiquer.
Dans le domaine de l'ingénierie du logiciel, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur de réduction des coûts et des délais.



Intérêt :

- Un support de raisonnements et de simulations.
- Une aide à l'élaboration et à la structuration des idées, c'est une bonne façon d'exprimer ses besoins.
- Un vecteur de communication entre personnes différentes.
- Une visualisation claire du système.
- En quelque sorte une documentation du système.

Techniques de modélisation :
Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements. Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, etc.) mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes :
– OMT (Object Modeling Technique) de James Rumbaugh (General Electric) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
– OOD (Object Oriented Design) de Grady Booch, définie pour le Département de la Défense, introduit le concept de paquetage (package) ;
– OOSE (Object Oriented Software Engeneering) d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou use cases).

2. 2. Concepts de modélisation

Systèmes, Modèles et Vues

- Une **Vue** d'un modèle de système représente un sous ensemble du modèle facilitant sa compréhension. C'est un concept très utilisé dans la décomposition ou le raffinement d'un modèle (Systèmes complexes). Les vues peuvent être imbriquées.

Exemple: vue statique, vue dynamique d'une application.

TD, TAD, et Instance*:*

- Un **Type de Donnée (TD)** est une abstraction définie dans le contexte d'un langage de programmation. Un TD a un nom qui dénote un ensemble de valeurs **instances** du TD) et définit la structure et les opérations valides pour toutes les instances du TD. Exemple: Int, Float.

- Un **Type Abstrait de Donnée (TAD)** est un TD défini par une spécification indépendante de son implémentation. Exemple: liste, ensemble, séquence. Un TAD peut avoir différentes implémentations.

Classes, Classes abstraite et Objets

- Une **Classe** est une abstraction dans la modélisation et la programmation OO. Elle encapsule une structure et un comportement. A la différence des TD, une classe peut avoir une relation d'héritage avec d'autres classes.
  Superclasse: classe de généralisation.
  Sous-classe: classe de spécialisation.
- Une **classe abstraite** est une classe de généralisation non instanciable. Exemple: collection en Java.
- Un **objet** est une instance d'une classe.

Classes événement, Evénement et messages

- Une **classe événement** est une abstraction d'événements pour lesquels le système fournit la même réponse.
- Un **événement** (occurrence particulière dans le système) est une instance d'une classe événement.
- Emission **message** est une classe événement. Chaque instance de message génère une instance de la classe événement (voir diagramme UML de séquence). Un message se compose d'un nom et d'arguments.

Prototype, Prototypage

- Un **prototypage** est l'action de raffinement de prototype.
- Un **prototype** est un modèle pouvant servir de base ou être un standard pour de futurs développements.

2. 3. UML

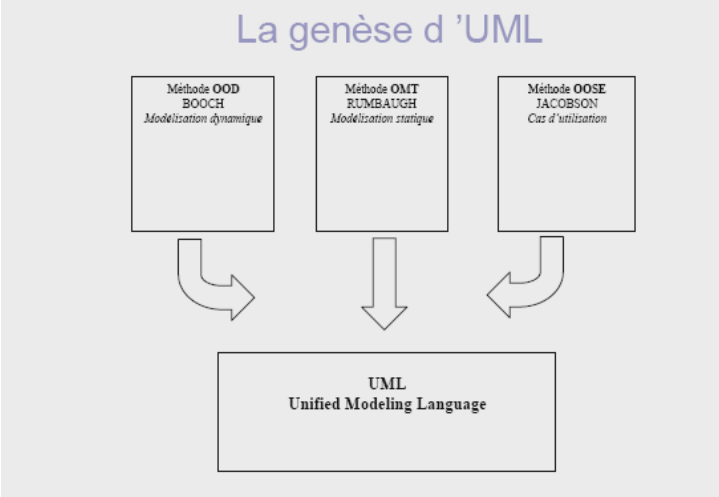
Un bon modèle devrait

- utiliser une notation standardisée;
- être compréhensible pour les clients et utilisateurs ;;
- permettre aux ingénieurs de logiciel de bien saisir le système ;
- procurer une vue abstraite du système ;
- être visuelle.

Pourquoi utiliser la modélisation visuelle?

- On doit enregistrer nos pensées et communiquer en utilisant des langages visuels et schématiques.
- On estime qu'au moins 50% de notre cerveau est impliqué dans le processus visuel.
- Les langages visuels sont naturels et faciles pour notre cerveau.

D'où :



Vers un langage unifié pour la modélisation

Chaque technique de modélisation orientée objet avait ses avantages et ses partisans. Le nombre de méthodes en compétition s'était réduit. Événement considérable et presque miraculeux, les trois gourous (Rumbaugh, Booch, et Jacobson) qui régnaient chacun sur l'une des trois méthodes OMT, OOD et OOSE se mirent d'accord pour définir une technique commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »). **UML** (Unified Modeling Language) est né de cet effort de convergence. L'adjectif unfied est là pour marquer qu'UML unifie, et donc remplace. En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage.